TASK INFERENCE BEYOND LEAST SQUARES IN BEHAVIORAL FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Behavioral Foundation Models (BFMs) aim to solve a wide range of downstream tasks without test-time policy learning by inferring a task vector from the reward function. While efficient, the retrieved policies are often suboptimal because of how this task vector is inferred, typically with ordinary least squares (OLS). OLS minimizes reward reconstruction error but leaves the ordering of rewards unconstrained, which can bias the successor measure of the retrieved zero-shot policy away from that of the optimal policy. In this work, we propose BLS, an efficient test-time inference method that balances minimizing reward reconstruction error with reducing successor-measure mismatch. Theoretically, we provide a suboptimality gap upper bound characterized by both successor-measure and reward-function residuals. Empirically, we evaluate BLS on top of state-of-the-art BFMs across benchmarks for locomotion, manipulation, and humanoid control. BLS outperforms existing task inference baselines with negligible computational overhead.

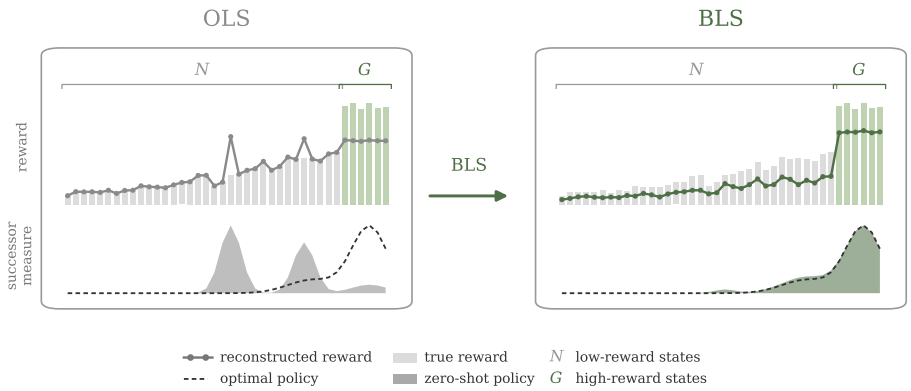

Figure 1: Ordinary least squares (OLS) aims to minimize the total errors of reconstructed rewards. Its performance highly depends on the quality of state representations, the state coverage within the dataset, and the distribution of rewards. Because OLS can be biased toward the majority of states, low-reward states (N) may be assigned with high-value rewards and vice versa for high-reward states (G). The successor measure of the retrieved zero-shot policy can therefore diverge from that of the optimal policy. BLS instead adopts an objective that reduces the reward reconstruction errors while enlarging the margin between high- and low-reward states. Thus, the successor measure of the retrieved zero-shot policy is encouraged to stay closer to the optimal policy's.

1 INTRODUCTION

Unsupervised zero-shot reinforcement learning (zero-shot RL; Touati et al., 2023) aims to offer a more general behavior learning framework than typical task-specific RL. During training, it does not require downstream task-specific reward functions, instead learning a repertoire of policies directly from interactions within the environment. At test time, given a downstream reward function, it can immediately retrieve a corresponding policy without additional policy learning or planning. Models trained via zero-shot RL can jointly handle downstream tasks in diverse forms, such as tracking reference trajectories, reaching goals, optimizing rewards, or following semantic instructions (Tirinzoni et al., 2025; Sikchi et al., 2024). Therefore, these models are often termed *Behavioral Foundation Models* (BFMs; Pirotta et al., 2024).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To empower zero-shot capabilities, BFMs rely on a linear formulation that connects arbitrary tasks with their reward functions. A BFM represents every state and *all possible* tasks as vectors in a joint latent space, where their inner products define linear reward functions (Barreto et al., 2017). It then trains policies conditioned on each task vector using the associated reward. Given any reward function at test time, the BFM directly obtains the corresponding policy by inferring the task vector through linear regression, with ordinary least squares (OLS) being the most commonly used solver in prior work (Bagatella et al., 2026a).

Although OLS minimizes reconstruction errors of a reward function among all states, it does not guarantee that the retrieved zero-shot policy is the best-performing one for the reward. The performance of a zero-shot policy heavily depends on the quality of state representations, the coverage of states within the dataset, and the distribution of rewards per task (Sikchi et al., 2025). Figure 1 presents an example: when rewards are dense but the highest-reward states are sparse, OLS is biased toward the majority of states, which are low-reward states in this example. This can leave the reconstruction of high-value rewards inaccurate, causing the retrieved zero-shot policy to become suboptimal. To tackle this, Sikchi et al. (2025) propose a fast adaptation strategy that minimizes the residual errors of OLS reconstruction but needs task-specific online interactions. Asadulaev et al. (2026) instead search for a task vector that maximizes rewards, while constraining the retrieved zero-shot policy to follow trajectories within the dataset, which are not optimal for the downstream task. Current methods remain restricted.

We propose BLS, a simple yet effective offline zero-shot inference framework Beyond Least Squares for BFMs. We first identify two sources of error that cause the suboptimality of zero-shot policies: drift in reward reconstruction and drift in state visitation distributions. While the former is covered by OLS, the latter is broadly overlooked in existing approaches. We then make two key observations of the optimal policy's state visitation distributions: (1) high-reward states are more frequently visited than low-reward states, (2) fixing the relative ranking of rewards also helps to preserve the optimal policy's behavior. To achieve both objectives, we propose a soft-margin contrastive loss that enlarges the margin in reconstructed rewards between high- and low-value original rewards. Finally, BLS jointly optimizes the task vector using the soft-margin loss and a trust-region loss with respect to OLS's solution. With the updated task vector, BLS aims to retrieve a zero-shot policy closer to the optimal policy than the one found by OLS.

We validate BLS with extensive experiments across several established RL benchmarks. On OGBench, BLS improves the average performance of all five feature representations under both proprioceptive and pixel-based observations. On DMC and HumEnv, BLS improves 18 out of 20 and 42 out of 45 dense-reward tasks, respectively. In particular, we find that BLS remains effective across different feature representations, reward structures, and task vector anchors. Furthermore, BLS incurs negligible computational overhead compared to OLS while bringing substantial performance gains. Finally, these results show that BLS is not merely an OLS correction but a general task-vector refinement framework for BFMs.

2 Related Work

2.1 Behavioral Foundation Models

Behavioral Foundation Models (BFMs) build on successor features Dayan (1993); Barreto et al. (2017) to learn a family of policies. Early approaches typically rely on separately learned state representations Wu et al. (2019); Ghosh et al. (2023); Park et al. (2024) and approximate the successor features induced by these fixed representations. Later, Touati and Ollivier (2021); Blier et al. (2021); Touati et al. (2023) introduce forward-backward representations that jointly learn the state features and successor measure. This approach achieves state-of-the-art performance on classical locomotion and manipulation tasks and has also been extended to humanoid control Tirinzoni et al. (2025); Li et al. (2026). Recent work further explores applications including zero-shot imitation learning Pirotta et al. (2024), general utility functions Bagatella et al. (2026b), and preference-based RL Yang et al. (2026). To better model successor measures, TD-JEPA Bagatella et al. (2026a) learns latent predictive representations that capture long-term dynamics, while Farebrother et al. (2025) uses flow matching to approximate the successor measure. Rather than modifying how representations and successor measures are trained, we optimize the inferred task vector z while keeping the representation, successor measure, and policy fixed.

2.2 INFERRING THE TASK VECTOR

In prior work on BFMs, ordinary least squares (OLS) is the most commonly used method to infer the task vector. To mitigate the reward projection error, ReLA Sikchi et al. (2025) proposes an off-policy actor-critic method that trains a small residual critic to compensate for the reward projection error while updating only the task vector z. LoLA Sikchi et al. (2025) is an actor-only alternative that optimizes a distribution over z from fixed-horizon on-policy rollouts bootstrapped by the frozen pretrained critic. Other works instead study task inference under different settings. OpTI-BFM Rupf et al. (2026) considers a practical setting in which reward labels are obtained only through environment interaction. It maintains uncertainty over the reward function from observed feature–reward pairs and selects z optimistically within the resulting confidence set, casting task inference as a linear bandit problem. ZOL Asadulaev et al. (2026) connects successor measures to stationary density ratios, showing that forward–backward representations implicitly encode the importance sampling weights between the policy- and data-induced occupancies. It uses these weights to perform distribution correction on the offline task inference objective, without additional training or interaction.

These works either operate within the learned feature basis or use online interaction to compensate for residual error. In contrast, BLS performs offline test-time task-vector refinement directly from reward-feature samples, using a surrogate objective motivated by our suboptimality gap.

3 PRELIMINARIES

We consider a reward-free Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \gamma, \rho)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(\cdot \mid s, a)$ is the transition kernel, $\gamma \in (0, 1)$ is the discount factor, and ρ is the initial state distribution. A downstream task is specified by a bounded reward function $r : \mathcal{S} \to \mathbb{R}$. An agent interacts with the MDP according to a policy $\pi : \mathcal{S} \to \mathrm{Prob}(\mathcal{A})$. Let $V_r^\pi = \mathbb{E}_{s_0\sim\rho, a_t\sim\pi(\cdot|s_t), s_{t+1}\sim P(\cdot|s_t,a_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1})\right]$ denote the expected cumulative discounted return, or *value*, of a policy π under a reward function r. RL aims to find an optimal policy π_r that maximizes the value: $\pi_r = \arg\max_\pi V_r^\pi$.

Successor measure. Executing a policy π in the MDP induces stochastic trajectories, which can be described by the discounted visitation measure over future states, also termed the *successor measure* Dayan (1993). Formally, it is written as: $M^\pi(s^+ \mid s, a) = \sum_{t=0}^{\infty} \gamma^t \Pr(s_{t+1} = s^+ \mid s, a; \pi)$, where $\Pr(s_{t+1} \mid s, a; \pi)$ is the probability of visiting state s_{t+1} at time step $t + 1$ starting from initial state-action pair (s, a), following the environment dynamics $s_{t+1} \sim P(\cdot|s_t, a_t)$ and the policy $a_t \sim \pi(\cdot|s_t)$. With the successor measure, the value V_r^π and the state-action value function $Q_r^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of a policy π under a reward function r can be written as $V_r^\pi = \mathbb{E}_{s_0\sim\rho, a_0\sim\pi(\cdot|s_0)} \left[\sum_{s^+\in\mathcal{S}} M^\pi(s^+ \mid s_0, a_0) r(s^+)\right]$ and $Q_r^\pi(s, a) = \sum_{s^+\in\mathcal{S}} M^\pi(s^+ \mid s, a) r(s^+)$.

Zero-shot RL and behavior foundation models. Unlike typical RL that learns an optimal policy based on a reward function specifying a task, zero-shot RL seeks to optimize policies for *all possible* tasks without any reward function given during training. To eliminate the need for handcrafting rewards, zero-shot RL adopts a linear form that associates every reward function with state features (Touati and Ollivier, 2021): $r_z(s) = \phi(s)^\top z$, where $\phi : \mathcal{S} \to \mathbb{R}^d$ is the state feature map, and $z \in \mathbb{R}^d$ denotes the task vector characterized by the reward function r_z. This formulation automatically constructs reward functions representing arbitrary tasks with randomly sampled task vectors. To address challenges in learning policies for every task, zero-shot RL combines the linear formulation with the successor measure, resulting in a self-supervised learning scheme (Touati et al., 2023):

$$\begin{cases} M^{\pi_z}(\cdot \mid s, a) = P(\cdot \mid s, a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a),\, a'\sim\pi_z(\cdot|s')} \left[M^{\pi_z}(\cdot \mid s', a')\right], \\ Q_{r_z}^{\pi_z}(s, a) = \sum_{s^+\in\mathcal{S}} M^{\pi_z}(s^+ \mid s, a)\, r_z(s^+) = \sum_{s^+\in\mathcal{S}} M^{\pi_z}(s^+ \mid s, a)\, \phi(s^+)^\top z, \\ \pi_z(s) = \operatorname*{argmax}_a Q_{r_z}^{\pi_z}(s, a). \end{cases} \tag{1}$$

where π_z denotes the policy conditioned on the task vector z. In other words, for each z and the policy π_z, zero-shot RL trains to model the policy's successor measure M^{π_z} with the Bellman equation, while optimizing the policy to maximize state-action values $Q_{r_z}^{\pi_z}$ estimated with the successor measure

and the reward r_z. At inference, given any reward function r, the task vector is calculated via OLS:

$$\hat{z}_{\mathrm{OLS}} = \operatorname*{argmin}_z \mathbb{E}_s \left[\|\phi(s)^\top z - r(s)\|_2^2 \right]. \tag{2}$$

With $\hat{z}_{\mathrm{OLS}}$, we retrieve the policy $\pi_{\hat{z}_{\mathrm{OLS}}}$ for the task represented by the reward function. Because we can directly obtain policies for any task (reward functions) with no extra training or planning, these models are often referred to as BFMs. Notably, if the reward function r lies in the span of the learned features ϕ, OLS exactly reconstructs r, and the retrieved zero-shot policy $\pi_{\hat{z}_{\mathrm{OLS}}}$ is optimal for the task (Ollivier, 2025). However, arbitrary downstream rewards may not lie in this feature span in practice, leading to non-negligible reconstruction errors and degraded task performance.

4 Beyond Least Squares

BLS is an effective zero-shot inference framework for BFMs, aiming to retrieve a better-performing policy than OLS for any reward function given at test time. We first analyze the value discrepancy, or *suboptimality gap*, between the optimal policy and a policy corresponding to a reconstructed reward function. We show that the suboptimality gap depends on both the reward reconstruction error and the difference between the two policies' successor measures, each of which motivates an objective for zero-shot inference. While the former coincides with OLS (Eqn. 2), the latter is intractable without knowing the optimal policy's successor measure in advance.

To bridge the gap, we gain insights from inductive biases of successor measures: optimal policies visit high-reward states more frequently than low-reward states, and reward functions sharing similar reward rankings tend to induce similar successor measures[1]. We propose a soft-margin contrastive loss to improve the reconstruction of high-value rewards and preserve the relative ranking of rewards. The task vector is optimized with both objectives via gradient descent.

We present the suboptimality gap analysis in Section 4.1 and the optimization objectives in Section 4.2.

4.1 Theoretical Analysis

We start by analyzing the suboptimality gap between the optimal policy and a policy retrieved by zero-shot inference. Let r and $\hat{r}$ denote the original and reconstructed reward functions. π_r and $\pi_{\hat{r}}$ are the optimal policies for r and $\hat{r}$, respectively. The suboptimality gap between these two policies is defined as $\mathrm{SubOpt}(r, \hat{r}) \coloneqq V_r^{\pi_r} - V_r^{\pi_{\hat{r}}}$. Next, we show that the upper bound on the suboptimality gap depends on both the reward reconstruction errors and the difference between the successor measures of π_r and $\pi_{\hat{r}}$.

Theorem 4.1 (Suboptimality gap)**.** *Let π_r and $\pi_{\hat{r}}$ be the optimal policies for rewards r and $\hat{r}$, respectively. Define the policy-averaged successor measure as $\bar{M}^\pi(s^+ \mid s) \coloneqq \mathbb{E}_{a\sim\pi(\cdot|s)}[M^\pi(s^+ \mid s, a)]$. We define $\Delta M(s^+ \mid s) \coloneqq \bar{M}^{\pi_{\hat{r}}}(s^+ \mid s) - \bar{M}^{\pi_r}(s^+ \mid s)$ as the residual between the two successor measures, and $\Delta r(s) \coloneqq \hat{r}(s) - r(s)$ as the residual between the two rewards. Then, the suboptimality gap $\mathrm{SubOpt}(r, \hat{r})$ of $\hat{r}$ has an upper bound:*

$$\mathrm{SubOpt}(r, \hat{r}) \;\le\; \|\Delta M\|_1 \;\cdot\; \|\Delta r\|_\infty.$$

The proof is in Appendix A. Theorem 4.1 reveals two conditions under which $\pi_{\hat{r}}$ achieves the same value as π_r under r. The first occurs when the reconstructed reward function $\hat{r}$ is the same as the original reward function r, and the second holds when $\pi_{\hat{r}}$'s policy-averaged successor measure matches that of π_r, even if the reconstructed rewards are incorrect. Notably, OLS pursues only the first condition while ignoring the second one. Because OLS in practice cannot reconstruct exact rewards, the residual of rewards can change the zero-shot policy's behavior and enlarge the residual of successor measures. The policy can therefore be suboptimal. In contrast, our key insight is to consider these conditions jointly, seeking the task vector that minimizes both the residual of rewards and successor measures.

[1]This is an empirical observation rather than a theoretical guarantee, though we find it holds in most practical settings.

4.2 PRACTICAL ALGORITHM

Devising an objective that explicitly minimizes the residual of successor measures remains a substantial challenge, as the optimal policy's successor measure is typically unknown at test time. We therefore investigate a surrogate objective based on inductive biases. We observe that high-density regions of a successor measure often correspond to high-reward states, while lowering (raising) rewards at high-(low-) reward states may significantly change the policy's behavior and also its successor measure. Based on these observations, we conjecture that preserving both high-value rewards and relative ranking of rewards is an effective and efficient objective to achieve the optimal successor measure. This motivates a soft-margin contrastive loss.

We propose an iterative optimization algorithm. Starting with OLS to find a task vector, the algorithm optimizes the vector via gradient descent, using a soft-margin contrastive loss and a trust-region loss. The former aims to reduce the residual of successor measures, and the latter constrains the task vector to stay close to OLS's solution, which minimizes the total reward reconstruction errors. We summarize the algorithm in Algorithm 1.

Soft-margin contrastive loss. We separate all states into two sets: high-reward states G and low-reward states N, with a threshold τ. The core idea is to enlarge the margins of reconstructed rewards between these two sets. However, a hard-margin contrastive loss $\mathcal{L}(z) = \max_{s\in N} \phi(s)^\top z - \min_{s\in G} \phi(s)^\top z$ is limited. It considers only one margin between the lowest reconstructed reward from G and the highest from N, ignoring other violating margins. Moreover, the loss contains non-smooth operations and is therefore less suitable for gradient-based optimization. To address this, we propose a soft-margin contrastive loss:

$$\mathcal{L}_{sm}(z) = \log \sum_{s\in G} \exp\big(-\phi(s)^\top z\big) + \log \sum_{s\in N} \exp\big(\phi(s)^\top z\big). \tag{3}$$

Intuitively, the first term of $\mathcal{L}_{sm}$ pushes up the bottom reconstructed rewards from G, while the second term drives down the top ones from N. The relative ranking among rewards is encouraged.

Trust-region loss. The soft-margin contrastive loss alone does not explicitly control the reward reconstruction error. We additionally introduce a trust-region loss: it regularizes the task vector z to stay close to OLS's solution, which minimizes the total reconstruction errors. The trust-region loss is written as $\mathcal{R}(z) = \mathbb{E}_s\left[\left\|\phi(s)^\top z - \phi(s)^\top \hat{z}_{\mathrm{OLS}}\right\|_2^2\right]$. Notably, this loss penalizes the discrepancy between rewards reconstructed from z and $\hat{z}_{\mathrm{OLS}}$, rather than the difference between the task vectors themselves. This design is better aligned with the objective of reducing the residual of rewards.

Finally, the total loss combines the soft-margin contrastive loss and the trust-region loss: $\mathcal{L}(z) = \mathcal{L}_{sm}(z) + \lambda\mathcal{R}(z)$ with a coefficient λ balancing these two losses. In our experiments, we show that the choice of λ correlates with the type of reward function: the trust-region loss (or OLS) benefits dense rewards, while the soft-margin contrastive loss is more helpful to sparse rewards. We further analyze the effect of λ in Section 5.5.

Algorithm 1 BLS

Require: a reward function r, a state feature map ϕ, threshold τ, number of iterations T, coefficient λ, and learning rate α.
Ensure: z
$G \leftarrow \{s : r(s) \geq \tau\}$
$N \leftarrow \{s : r(s) < \tau\}$
$\hat{z}_{\mathrm{OLS}} \leftarrow \arg\min_z \mathbb{E}_s\left[\|\phi(s)^\top z - r(s)\|_2^2\right]$
$z \leftarrow \hat{z}_{\mathrm{OLS}}$
for $t = 1 \dots T$ **do**
  $\mathcal{L}_{sm}(z) \leftarrow \log \sum_{s\in G} \exp\left(-\phi(s)^\top z\right) + \log \sum_{s\in N} \exp\left(\phi(s)^\top z\right)$
  $\mathcal{R}(z) \leftarrow \mathbb{E}_s\left[\left\|\phi(s)^\top z - \phi(s)^\top \hat{z}_{\mathrm{OLS}}\right\|_2^2\right]$
  $z \leftarrow z - \alpha\nabla_z\left(\mathcal{L}_{sm}(z) + \lambda\mathcal{R}(z)\right)$
  $z \leftarrow z/\|z\|_2$
return z

Table 1: Zero-shot success rates (%) on OGBench across five feature representations under proprioceptive and pixel-based observations. BLS improves the average performance of all representations.

	Laplacian		ICVF		HILP		FB		TD-JEPA	
	OLS	BLS	OLS	BLS	OLS	BLS	OLS	BLS	OLS	BLS
antmaze-mn	48.1 ± 2.9	**67.3** ± 8.1	66.3 ± 1.2	**69.2** ± 6.7	74.3 ± 1.4	**87.7** ± 1.5	77.5 ± 2.6	**88.8** ± 4.2	63.1 ± 0.8	**83.2** ± 7.7
antmaze-ln	7.6 ± 1.8	**10.9** ± 2.9	56.0 ± 4.1	**65.5** ± 4.9	49.2 ± 2.1	**63.7** ± 1.8	52.4 ± 4.0	**63.5** ± 9.2	50.7 ± 2.4	**64.3** ± 7.0
antmaze-ms	15.1 ± 7.8	**34.7** ± 3.4	28.8 ± 5.6	**36.5** ± 9.1	26.3 ± 9.1	**31.3** ± 2.3	69.9 ± 1.7	**79.6** ± 2.1	54.5 ± 4.8	**59.5** ± 2.4
antmaze-ls	15.2 ± 3.9	**15.3** ± 0.6	5.5 ± 0.8	4.0 ± 0.7	11.9 ± 1.2	**13.5** ± 0.8	51.5 ± 2.9	45.7 ± 4.6	39.2 ± 5.9	**45.1** ± 5.9
antmaze-me	3.7 ± 3.3	**7.6** ± 1.7	0.3 ± 0.5	0.0 ± 0.0	2.4 ± 2.6	**6.7** ± 1.5	50.3 ± 2.0	49.2 ± 4.9	15.1 ± 5.5	**29.2** ± 7.0
cube-single	19.9 ± 1.3	18.3 ± 4.6	19.5 ± 1.2	13.3 ± 0.5	84.5 ± 5.7	82.0 ± 6.7	50.4 ± 0.8	**54.0** ± 1.8	20.3 ± 2.9	**55.9** ± 3.0
cube-double	1.3 ± 0.6	**2.0** ± 1.8	3.1 ± 1.5	**3.5** ± 0.8	2.8 ± 3.1	**25.6** ± 2.1	1.9 ± 1.0	**3.1** ± 1.2	1.7 ± 0.5	**2.1** ± 1.2
scene	4.3 ± 1.4	4.0 ± 1.7	39.3 ± 3.2	**47.7** ± 3.7	26.3 ± 2.7	**36.8** ± 2.2	7.6 ± 0.7	**9.6** ± 0.4	37.9 ± 3.2	**54.0** ± 6.8
puzzle-3×3	1.5 ± 0.6	**1.6** ± 0.8	13.6 ± 0.7	**15.3** ± 1.4	2.4 ± 0.0	**4.5** ± 1.5	5.7 ± 1.2	**9.6** ± 1.8	11.2 ± 0.4	**14.1** ± 2.1
Average	13.0 ± 1.1	**18.0** ± 1.2	25.8 ± 0.9	**28.3** ± 1.4	31.1 ± 1.4	**39.1** ± 0.9	40.8 ± 0.7	**44.8** ± 1.4	32.6 ± 1.2	**45.3** ± 1.8
vis-antmaze-mn	96.7 ± 0.8	95.5 ± 3.5	85.9 ± 1.5	**89.3** ± 1.6	82.3 ± 3.7	**93.3** ± 0.6	96.7 ± 0.8	95.3 ± 1.3	75.5 ± 3.8	**88.8** ± 2.4
vis-antmaze-ln	14.7 ± 5.7	**20.7** ± 1.2	45.6 ± 2.0	42.0 ± 4.2	65.9 ± 3.5	**79.2** ± 5.6	80.0 ± 2.5	77.9 ± 1.6	61.6 ± 2.6	**66.9** ± 6.1
vis-antmaze-ms	46.5 ± 1.5	**66.3** ± 1.2	31.5 ± 0.8	**32.0** ± 5.6	62.9 ± 2.8	**63.5** ± 1.7	80.1 ± 0.8	80.0 ± 1.1	74.9 ± 2.4	**76.1** ± 5.5
vis-antmaze-ls	1.5 ± 1.3	**2.3** ± 2.2	6.9 ± 3.8	**7.3** ± 1.1	9.7 ± 4.8	**16.4** ± 5.6	10.5 ± 4.4	10.0 ± 6.6	31.3 ± 2.1	**34.1** ± 0.8
vis-antmaze-me	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
vis-cube-single	45.7 ± 9.8	**50.9** ± 2.1	35.9 ± 3.6	**36.0** ± 3.7	66.3 ± 2.0	**68.7** ± 3.0	47.9 ± 2.7	**56.7** ± 4.6	47.7 ± 3.6	**52.4** ± 4.3
vis-cube-double	0.1 ± 0.2	**3.3** ± 5.8	0.0 ± 0.0	0.0 ± 0.0	2.5 ± 0.6	2.5 ± 1.7	0.9 ± 0.5	0.5 ± 0.5	0.5 ± 0.5	**3.5** ± 1.6
vis-scene	2.8 ± 0.7	**16.3** ± 9.9	2.8 ± 1.4	**3.1** ± 0.5	8.7 ± 1.7	**28.5** ± 4.3	14.5 ± 2.2	**25.3** ± 3.8	28.9 ± 2.7	**34.9** ± 2.8
vis-puzzle-3×3	12.4 ± 3.0	**17.1** ± 1.5	0.0 ± 0.0	0.0 ± 0.0	2.4 ± 0.4	**2.5** ± 1.2	7.2 ± 1.1	6.3 ± 2.3	0.8 ± 0.4	**1.6** ± 1.4
Average	24.5 ± 1.3	**30.3** ± 1.4	23.2 ± 0.7	**23.3** ± 0.9	33.4 ± 0.9	**39.4** ± 1.1	37.5 ± 0.7	**39.1** ± 1.1	35.7 ± 0.8	**39.8** ± 1.1

5 EXPERIMENTS

We systematically evaluate BLS on established RL simulation benchmarks across a wide range of tasks, including 5 navigation and 4 manipulation tasks on OGBench (Park et al., 2025), 20 continuous-control tasks across 4 domains on ExoRL/DMC (Tassa et al., 2018; Yarats et al., 2022), and 45 motion-tracking tasks with a humanoid robot on HumEnv (Tirinzoni et al., 2025). We evaluate BLS along five main aspects. First, we examine its robustness to different feature representations. Second, we compare BLS against existing zero-shot inference methods, including two offline approaches (OLS and ZOL (Asadulaev et al., 2026)) and two online adaptation methods (ReLA and LoLA (Sikchi et al., 2025)). Third, we compare BLS with goal-conditioned RL. Fourth, we evaluate BLS separately on dense- and sparse-reward tasks. Finally, we conduct ablations and further analyses to examine its design choices, robustness, and computational efficiency. Unless otherwise stated, we report the mean and standard deviation over 3 random seeds. Additional experimental results and analyses are provided in Appendix C.

5.1 IS BLS ROBUST TO DIFFERENT CHOICES OF FEATURE REPRESENTATIONS?

We evaluate BLS on OGBench across 5 feature representations: Laplacian (Wu et al., 2019), ICVF (Ghosh et al., 2023), HILP (Park et al., 2024), FB (Touati and Ollivier, 2021), and TD-JEPA (Bagatella et al., 2026a). These representations are pretrained with different objectives. OGBench contains 9 sparse-reward navigation and manipulation tasks. We consider both proprioceptive and pixel-based observations. For each representation and observation modality, we use a fixed λ across all tasks; full configuration details are provided in Appendix B.

As shown in Table 1, BLS improves the average performance of all feature representations under both observation modalities. Under proprioceptive observations, the gains are particularly large for HILP ($31.1\% \rightarrow 39.1\%$) and TD-JEPA ($32.6\% \rightarrow 45.3\%$). The same overall trend holds for pixel-based observations, showing that BLS is robust across both feature representations and observation settings.

Table 2: Average results over FB on the OGBench comparing BLS against offline baseline (ZOL) from different task anchor (OLS/ZOL).

	OLS	ZOL	BLS@OLS	BLS@ZOL
antmaze-mn	77.5 ± 2.6	84.5 ± 1.6	**88.8** ± 4.2	84.4 ± 4.5
antmaze-ln	52.4 ± 4.0	41.5 ± 0.8	63.5 ± 9.2	**68.7** ± 2.4
antmaze-ms	69.9 ± 1.7	51.9 ± 1.2	**79.6** ± 2.1	68.1 ± 3.6
antmaze-ls	**51.5** ± 2.9	37.9 ± 5.1	45.7 ± 4.6	36.0 ± 4.6
antmaze-me	**50.3** ± 2.0	49.7 ± 3.1	49.2 ± 4.9	48.8 ± 1.4
cube-single	50.4 ± 0.8	46.4 ± 6.8	**54.0** ± 1.8	43.7 ± 6.5
cube-double	1.9 ± 1.0	2.8 ± 0.0	**3.1** ± 1.2	1.7 ± 1.0
scene	7.6 ± 0.7	33.5 ± 7.9	9.6 ± 0.4	**35.3** ± 8.4
puzzle-3×3	5.7 ± 1.2	**13.7** ± 1.2	9.6 ± 1.8	11.7 ± 1.6
Average	40.8 ± 0.7	40.2 ± 1.4	**44.8** ± 1.4	44.3 ± 1.5

Table 3: Average results over five representations on OGBench comparing offline OLS / BLS against online ReLA/LoLA.

	OLS	ReLA	LoLA	BLS
antmaze-mn	65.9 ± 0.9	49.7 ± 2.1	64.9 ± 2.4	**79.2** ± 2.8
antmaze-ln	43.2 ± 1.4	19.4 ± 2.5	36.4 ± 2.6	**53.6** ± 2.6
antmaze-ms	38.9 ± 2.8	33.3 ± 2.8	39.5 ± 2.8	**48.3** ± 2.1
antmaze-ls	24.7 ± 1.6	8.3 ± 1.7	20.5 ± 3.0	**24.7** ± 1.5
antmaze-me	14.4 ± 1.4	10.7 ± 1.5	16.2 ± 2.3	**18.5** ± 1.8
cube-single	38.9 ± 1.3	38.5 ± 1.2	40.6 ± 1.5	**44.7** ± 1.8
cube-double	2.2 ± 0.7	2.7 ± 0.5	2.3 ± 0.6	**7.3** ± 0.7
scene	23.1 ± 1.1	16.1 ± 2.8	21.7 ± 2.3	**30.4** ± 1.6
puzzle-3×3	6.9 ± 0.3	7.9 ± 1.1	7.0 ± 1.2	**9.0** ± 0.7
Average	28.7 ± 0.5	20.8 ± 0.7	27.7 ± 0.7	**35.1** ± 0.6

5.2 How does BLS compare with existing task inference methods?

We first compare BLS against two offline zero-shot inference methods, OLS and ZOL, on OGBench. Since ZOL is specific to the FB representation (Touati and Ollivier, 2021), we use FB for all methods in this comparison. The trust-region loss of BLS can use either OLS or ZOL as its anchor. We denote these variants as BLS@OLS and BLS@ZOL, respectively. As shown in Table 2, both variants improve their corresponding anchors on average. BLS@OLS increases the average success rate from 40.8% to 44.8%, while BLS@ZOL improves ZOL from 40.2% to 44.3%, corresponding to gains of 4.0 and 4.1 percentage points.

We next compare our offline inference method against the online adaptation methods ReLA and LoLA. ReLA and LoLA use 20 reward-labeled interaction episodes per task, whereas BLS task inference requires no task-specific interaction. We evaluate all methods over 5 feature representations: Laplacian, ICVF, HILP, FB, and TD-JEPA, and report the average performance across them; per-representation results are provided in Appendix C.1. As shown in Table 3, BLS achieves an average success rate of 35.1%, outperforming ReLA (20.8%) and LoLA (27.7%) by 14.3 and 7.4 percentage points, respectively.

5.3 How does BLS compare with goal-conditioned RL?

We report the success rate of BLS against goal-conditioned RL (GCRL) on the proprioceptive setting of OGBench. For a fair comparison, we use HILP for both zero-shot RL and GCRL, as HILP supports both formulations within the same representation framework. GCRL provides a strong reference point for goal-reaching tasks because it is trained directly on the goal-conditioned objective and receives the goal explicitly, whereas zero-shot RL must infer a task vector from the reward function. As shown in Figure 2, BLS achieves an average success rate of 39.1%, comparable to GCRL at 37.7%, while OLS reaches only 31.1%. The relative performance varies across tasks: BLS performs particularly well on several navigation tasks, whereas GCRL is stronger on some manipulation tasks such as *cube-double* and *scene*. These results show that improved task inference can substantially close the performance gap between zero-shot RL and direct goal-conditioned control.

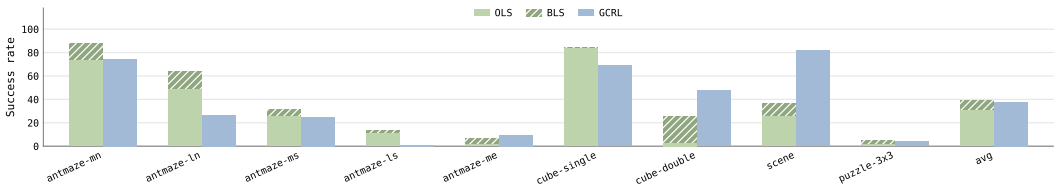

Figure 2: Comparison of OLS, BLS, and GCRL on OGBench using HILP. The hatched segments indicate the improvement of BLS over OLS. BLS achieves performance comparable to GCRL on average, while their relative performance varies across tasks.

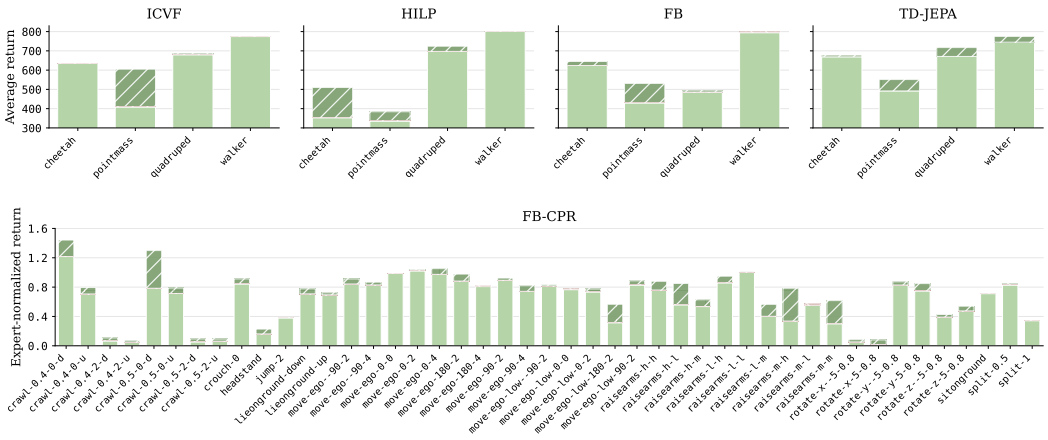

Figure 3: Dense-reward results on DMC (top) and HumEnv (bottom). Solid bars show the return of the OLS policy, and hatched segments indicate the change after applying BLS. DMC reports average return across tasks for each domain and feature representation, while HumEnv reports expert-normalized return on 45 motion-tracking tasks, with FB-CPR feature representations.

5.4 How does BLS perform on dense-reward tasks?

We next investigate whether BLS remains effective on dense-reward tasks. We evaluate on DMC with 20 tasks across 4 domains and on HumEnv with 45 humanoid control tasks. Since dense rewards do not naturally define a sparse goal set, we construct G from the high-reward region of each benchmark. We define G using the top 5% high-reward region and let the remaining states form N. Full configuration and evaluation details are provided in Appendix B.

The results are summarized in Figure 3. Although the gains are generally smaller than on sparse-reward tasks, BLS remains effective on both benchmarks. On DMC, after averaging over the four feature representations (ICVF, HILP, FB, and TD-JEPA), BLS improves 18 of 20 tasks. On HumEnv, using the FB-CPR representation (Tirinzoni et al., 2025), it improves 42 of 45 tasks, with the largest gains on *crawl* and *raisearms*. Overall, these results show that BLS also generalizes effectively to dense-reward tasks across different control domains.

5.5 Ablations and Further Analysis

We further analyze BLS in terms of (1) sensitivity to λ, (2) robustness to the construction of the high- and low-reward sets G and N, (3) computational overhead, and (4) robustness to the feature dimension d.

The sensitivity to the choice of λ. We observed that small values of λ are suited for sparse-reward tasks, and vice versa. We conducted experiments on two sparse-reward tasks (*antmaze* and *scene*) from OGBench, and all tasks with the *walker* and *quadruped* embodiments from DMC. We report the performance obtained with each λ selected from the range $\{0, 0.1, 0.3, 1, 3, 10, 30, 100, 300\}$. Results are summarized in Figure 4. We show that keeping the task vector close to OLS's solution

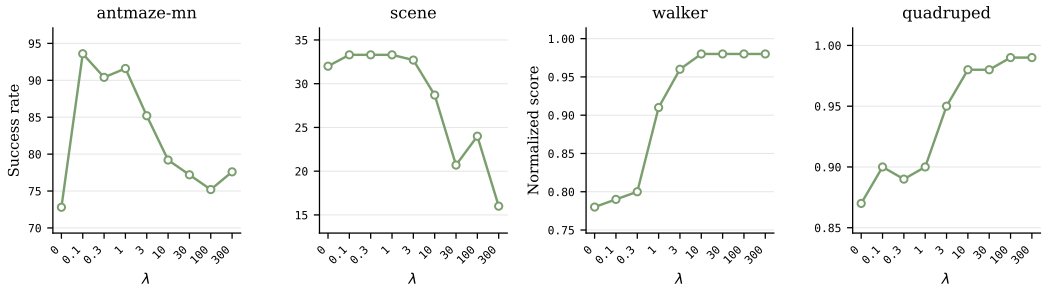

Figure 4: Sensitivity to λ with the HILP encoder: sparse rewards (left two) prefer a small λ, and dense rewards (right two) prefer a large one.

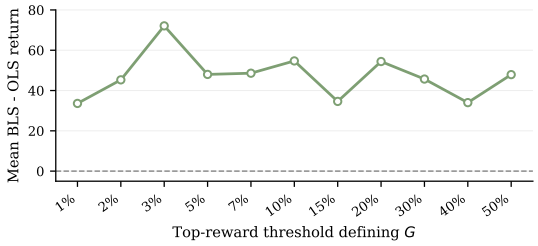

Figure 5: Sensitivity to the reward threshold for defining G on DMC with TD-JEPA. We report the paired mean return gain of BLS over OLS across 20 tasks as the threshold varies.

(larger λ) is fine for dense-reward tasks, but it soon becomes ineffective when the rewards get sparser. Enlarging the margins between low- and high-value rewards significantly improves the task performance under the sparse-reward regime.

Sensitivity to the reward threshold. For dense-reward tasks, we define G using a threshold on the reward distribution. We evaluate the sensitivity to this threshold on all 20 DMC tasks with TD-JEPA, varying the cutoff from the top 1% to the top 50% of reward samples. As shown in Figure 5, BLS achieves a positive paired mean improvement over OLS across the entire range. For thresholds between the top 3% and top 50%, the mean improvement ranges from 34 to 72 return points, showing substantial gains across threshold choices. Thus, BLS does not require careful tuning of the reward threshold used to construct G.

The computational efficiency. While BLS adopts gradient-based optimization, it incurs little overhead during inference. Using an Nvidia RTX 4090 GPU card, optimizing the task vector for 500 steps costs less than 1 second on OGBench and 1.4 seconds on HumEnv. Notably, the inference procedure is performed once before executing the task. Therefore, the incurred latency is negligible.

Table 4: Success rate as the feature dimension d varies.

d	25	50	100	200	500	1000
OLS	70.9 $\pm$ 2.1	74.3 $\pm$ 1.4	59.2 $\pm$ 1.7	25.3 $\pm$ 1.5	9.2 $\pm$ 0.4	22.0 $\pm$ 5.7
BLS	**84.7** $\pm$ 2.0	**87.7** $\pm$ 1.5	**87.5** $\pm$ 2.0	**82.1** $\pm$ 9.0	**79.6** $\pm$ 8.4	**82.7** $\pm$ 16.7

Robustness to feature dimension. A larger feature dimension d increases representation capacity, but does not guarantee a better learned representation. This distinction is important for OLS, whose effectiveness depends on how well the downstream reward can be represented by the learned feature basis. We test this by training HILP on *antmaze-mn* with $d \in \{25, 50, 100, 200, 500, 1000\}$ while keeping the number of reward samples fixed for task inference. As shown in Table 4, increasing d does not make OLS more reliable: its success rate peaks at 74.3% with $d = 50$ and then drops sharply to 9.2% at $d = 500$. In contrast, BLS outperforms OLS at every d, remaining between 79% and 88% across all dimensions. These results show that increasing feature capacity alone cannot resolve the limitation of OLS, whereas BLS remains effective even when the learned representation does not yield a reliable OLS solution.

6 Conclusion

In this work, we study the limitations of OLS-based task inference in Behavioral Foundation Models, showing that the quality of an inferred task vector depends not only on reward reconstruction error but also on the successor measure induced by the reconstructed reward. We propose BLS, which refines the task vector with a soft-margin objective while staying close to a given inference anchor, and consistently outperforms existing task inference methods with little additional computation. While these results are encouraging, the soft-margin objective remains an empirical surrogate for successor-measure mismatch, and its weight against the anchor depends on reward sparsity. We hope our results encourage future work to consider the induced policy behavior, rather than reward reconstruction alone, when designing task inference methods for BFMs.

AI USE STATEMENT

We used large language models to assist with checking the proof, implementing and debugging experimental code, interpreting results, and polishing the text. All ideas, methods, and claims originate from the authors, and every AI-assisted proof, code path, and passage was verified by us.

REPRODUCIBILITY STATEMENT

We provide the details necessary to reproduce our results throughout the paper and appendix. The BLS objective and optimization procedure are specified in Section 4.2 and Algorithm 1, with complete proofs provided in Appendix A. Appendix B reports the implementation and evaluation details, including task-inference data, optimization hyperparameters, benchmark-specific configurations, evaluation protocols, and random-seed settings. Additional task-level results and robustness analyses are provided in Appendix C.

REFERENCES

Arip Asadulaev, Maksim Bobrin, Salem Lahlou, Dmitry Dylov, Fakhri Karray, and Martin Takac. Zero-shot off-policy learning. In *International Conference on Machine Learning (ICML)*, 2026.

Marco Bagatella, Matteo Pirotta, Ahmed Touati, Alessandro Lazaric, and Andrea Tirinzoni. Td-jepa: Latent-predictive representations for zero-shot reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026a.

Marco Bagatella, Thomas Rupf, Georg Martius, and Andreas Krause. Soft forward-backward representations for zero-shot reinforcement learning with general utilities. In *Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning Workshop at ICML*, 2026b.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Léonard Blier, Corentin Tallec, and Yann Ollivier. Learning successor states and goal-dependent values: A mathematical viewpoint. *arXiv preprint arXiv:2101.07123*, 2021.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 1993.

Jesse Farebrother, Matteo Pirotta, Andrea Tirinzoni, Rémi Munos, Alessandro Lazaric, and Ahmed Touati. Temporal difference flows. In *International Conference on Machine Learning (ICML)*, 2025.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Reinforcement learning from passive data via latent intentions. In *International Conference on Machine Learning (ICML)*, 2023.

Yitang Li, Zhengyi Luo, Tonghe Zhang, Cunxi Dai, Anssi Kanervisto, Andrea Tirinzoni, Haoyang Weng, Kris Kitani, Mateusz Guzek, Ahmed Touati, et al. Bfm-zero: A promptable behavioral foundation model for humanoid control using unsupervised reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Yann Ollivier. Which features are best for successor features? *arXiv preprint arXiv:2502.10790*, 2025.

Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with hilbert representations. In *International Conference on Machine Learning (ICML)*, 2024.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations (ICLR)*, 2025.

Matteo Pirotta, Andrea Tirinzoni, Ahmed Touati, Alessandro Lazaric, and Yann Ollivier. Fast imitation via behavior foundation models. In *International Conference on Learning Representations (ICLR)*, 2024.

Thomas Rupf, Marco Bagatella, Marin Vlastelica, and Andreas Krause. Optimistic task inference for behavior foundation models. In *International Conference on Learning Representations (ICLR)*, 2026.

Harshit Sikchi, Siddhant Agarwal, Pranaya Jajoo, Samyak Parajuli, Caleb Chuck, Max Rudolph, Peter Stone, Amy Zhang, and Scott Niekum. Rl zero: Zero-shot language to behaviors without any supervision. *arXiv preprint arXiv:2412.05718*, 2024.

Harshit Sikchi, Andrea Tirinzoni, Ahmed Touati, Yingchen Xu, Anssi Kanervisto, Scott Niekum, Amy Zhang, Alessandro Lazaric, and Matteo Pirotta. Fast adaptation with behavioral foundation models. *arXiv preprint arXiv:2504.07896*, 2025.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Andrea Tirinzoni, Ahmed Touati, Jesse Farebrother, Mateusz Guzek, Anssi Kanervisto, Yingchen Xu, Alessandro Lazaric, and Matteo Pirotta. Zero-shot whole-body humanoid control via behavioral foundation models. In *International Conference on Learning Representations (ICLR)*, 2025.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *International Conference on Learning Representations (ICLR)*, 2023.

Yifan Wu, George Tucker, and Ofir Nachum. The laplacian in rl: Learning representations with efficient approximations. In *International Conference on Learning Representations (ICLR)*, 2019.

Jun-Jie Yang, Chia-Heng Hsu, Kui-Yuan Chen, and Ping-Chun Hsieh. From reward-free representations to preferences: Rethinking offline preference-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2026.

Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. In *Generalizable Policy Learning in the Physical World Workshop at ICLR*, 2022.

A PROOFS

This section provides the proof of Theorem 4.1. Throughout this section, $\mathcal{S}$ and $\mathcal{A}$ are finite and $\gamma \in (0, 1)$. Recall that ρ denotes the initial state distribution. For a policy π, we define its policy-averaged successor measure as $\bar{M}^\pi(s^+ \mid s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[M^\pi(s^+ \mid s, a)]$. For a kernel $K(\cdot \mid s)$ and a function f, we write $\langle K, f\rangle_\rho = \mathbb{E}_{s\sim\rho} \sum_{s^+} K(s^+ \mid s) f(s^+)$, and define the mixed norm $\|K\|_1 = \sup_s \sum_{s^+} |K(s^+ \mid s)|$. Under this notation, $V_r^\pi = \langle \bar{M}^\pi, r\rangle_\rho$.

We restate Theorem 4.1 for convenience.

Theorem A.1 (Suboptimality gap)**.** *Let π_r and $\pi_{\hat{r}}$ be the optimal policies for rewards r and $\hat{r}$, respectively, and let $\bar{M}^{\pi_r}$ and $\bar{M}^{\pi_{\hat{r}}}$ be their corresponding policy-averaged successor measures. Define $\Delta M(s^+ \mid s) \coloneqq \bar{M}^{\pi_{\hat{r}}}(s^+ \mid s) - \bar{M}^{\pi_r}(s^+ \mid s)$ as the residual between the two successor measures, and $\Delta r(s) \coloneqq \hat{r}(s) - r(s)$ as the residual between the two rewards. Then,*

$$\mathrm{SubOpt}(r, \hat{r}) \le \|\Delta M\|_1 \cdot \|\Delta r\|_\infty.$$

Proof. We first relate the suboptimality gap to ΔM and Δr. Since $\pi_{\hat{r}}$ is optimal for $\hat{r}$,

$$\langle \Delta M,\, \hat{r}\rangle_\rho = V_{\hat{r}}^{\pi_{\hat{r}}} - V_{\hat{r}}^{\pi_r} \ge 0 \tag{4}$$

. By Eqn. 4 and definition of ΔM and Δr, we have

$$\begin{aligned} \mathrm{SubOpt}(r, \hat{r}) &= -\langle \Delta M,\, r\rangle_\rho && (5)\\ &= \langle \Delta M,\, \Delta r\rangle_\rho - \langle \Delta M,\, \hat{r}\rangle_\rho && (6)\\ &\le \langle \Delta M,\, \Delta r\rangle_\rho && (7) \end{aligned}$$

Applying Hölder's inequality to each s gives

$$\begin{aligned} |\langle \Delta M, \Delta r\rangle_\rho| &\le \mathbb{E}_{s\sim\rho} \sum_{s^+\in\mathcal{S}} |\Delta M(s^+ \mid s)||\Delta r(s^+)| && (8)\\ &\le \|\Delta M\|_1 \|\Delta r\|_\infty. && (9) \end{aligned}$$

Combining Eqn. 7 and Eqn. 9, we conclude that

$$\mathrm{SubOpt}(r, \hat{r}) \le \|\Delta M\|_1 \|\Delta r\|_\infty.$$

□

B IMPLEMENTATION AND EVALUATION DETAILS

Common inference and optimization setup. For OGBench and DMC, we use $10{,}000$ transitions sampled from the offline replay buffer for task inference. We follow the sampling procedure of each pretrained representation. The corresponding next states are used to compute the feature representations and rewards, and the same inference batch is used for both OLS and BLS. This setting is used for both proprioceptive and pixel-based OGBench experiments. For HumEnv, we use $50{,}000$ inference samples. We optimize the task vector for $T = 500$ steps using Adam with a learning rate of $\alpha = 0.03$. After each gradient update, the task vector is projected onto the unit sphere. Reconstructed rewards are normalized by the standard deviation of the rewards reconstructed from the anchor task vector, and the normalized scores are used in the soft-margin objective.

Benchmark-specific configurations. On OGBench, G consists of states attaining the maximum reward, and the remaining states form N. For each feature representation and observation modality, we use a fixed λ across all tasks. For proprioceptive Laplacian, ICVF, HILP, FB, and TD-JEPA, we use $\lambda = 10, 2, 3, 0.3$, and 2, respectively. For their pixel-based counterparts, we use $\lambda = 2, 10, 2, 20$, and 3, respectively. For dense-reward tasks on DMC and HumEnv motion tracking, we define G as the top 5% of reward samples and let the remaining samples form N. Since the same λ can induce different deviations from the anchor across tasks, we parameterize the trust-region strength by the cosine similarity between the refined and anchor task vectors. We use a fixed cosine similarity target of 0.95 for all such experiments, where λ is chosen to match this target between the refined and anchor task vectors. For HumEnv goal-reaching, an explicit reward is not provided. We therefore define G as the 100 states in the offline buffer closest to the target pose and use the benchmark proximity function as the surrogate reward. We use a fixed $\lambda = 20$ for all target poses.

Table 5: OGBench success rates (%) across four feature representations.

Laplacian				
Task	OLS	BLS	ReLA	LoLA
antmaze-mn	48.1 ± 2.9	**67.3** ± 8.1	39.6 ± 8.0	48.5 ± 9.2
antmaze-ln	7.6 ± 1.8	**10.9** ± 2.9	7.3 ± 6.1	5.6 ± 1.1
antmaze-ms	15.1 ± 7.8	**34.7** ± 3.4	12.9 ± 8.5	10.8 ± 4.2
antmaze-ls	15.2 ± 3.9	**15.3** ± 0.6	4.9 ± 5.8	12.9 ± 5.7
antmaze-me	3.7 ± 3.3	**7.6** ± 1.7	0.0 ± 0.0	3.5 ± 3.6
cube-single	19.9 ± 1.3	18.3 ± 4.6	18.5 ± 3.4	**23.6** ± 1.4
cube-double	1.3 ± 0.6	2.0 ± 1.8	**2.4** ± 0.7	1.9 ± 0.8
scene	4.3 ± 1.4	4.0 ± 1.7	4.3 ± 1.2	**5.3** ± 0.8
puzzle-3×3	1.5 ± 0.6	1.6 ± 0.8	1.7 ± 0.6	**3.5** ± 0.6
Average	13.0	**18.0**	10.2	12.8

ICVF				
Task	OLS	BLS	ReLA	LoLA
antmaze-mn	66.3 ± 1.2	**69.2** ± 6.7	51.5 ± 3.2	53.6 ± 2.4
antmaze-ln	56.0 ± 4.1	**65.5** ± 4.9	25.6 ± 10.0	39.5 ± 5.4
antmaze-ms	28.8 ± 5.6	**36.5** ± 9.1	18.3 ± 4.0	33.5 ± 5.4
antmaze-ls	5.5 ± 0.8	4.0 ± 0.7	**6.8** ± 6.2	**6.8** ± 2.9
antmaze-me	**0.3** ± 0.5	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
cube-single	**19.5** ± 1.2	13.3 ± 0.5	13.7 ± 0.6	19.3 ± 2.0
cube-double	3.1 ± 1.5	3.5 ± 0.8	2.8 ± 1.1	**3.6** ± 1.6
scene	39.3 ± 3.2	**47.7** ± 3.7	37.2 ± 2.2	37.5 ± 5.9
puzzle-3×3	13.6 ± 0.7	**15.3** ± 1.4	8.7 ± 1.6	12.4 ± 3.9
Average	25.8	**28.3**	18.3	22.9

HILP				
Task	OLS	BLS	ReLA	LoLA
antmaze-mn	74.3 ± 1.4	**87.7** ± 1.5	65.9 ± 2.0	76.7 ± 3.0
antmaze-ln	49.2 ± 2.1	**63.7** ± 1.8	24.5 ± 2.9	42.5 ± 9.5
antmaze-ms	26.3 ± 9.1	**31.3** ± 2.3	12.7 ± 9.8	23.7 ± 11.5
antmaze-ls	11.9 ± 1.2	**13.5** ± 0.8	0.7 ± 0.6	4.9 ± 6.9
antmaze-me	2.4 ± 2.6	6.7 ± 1.5	**8.0** ± 7.5	7.3 ± 5.3
cube-single	**84.5** ± 5.7	82.0 ± 6.7	**84.5** ± 3.5	83.9 ± 6.0
cube-double	2.8 ± 3.1	**25.6** ± 2.1	1.3 ± 2.0	2.9 ± 2.0
scene	26.3 ± 2.7	**36.8** ± 2.2	12.5 ± 10.9	16.9 ± 9.4
puzzle-3×3	2.4 ± 0.0	**4.5** ± 1.5	3.1 ± 2.7	3.2 ± 0.4
Average	31.1	**39.1**	23.7	29.1

TD-JEPA				
Task	OLS	BLS	ReLA	LoLA
antmaze-mn	63.1 ± 0.8	**83.2** ± 7.7	20.5 ± 5.1	65.2 ± 6.6
antmaze-ln	50.7 ± 2.4	**64.3** ± 7.0	0.0 ± 0.0	41.7 ± 3.4
antmaze-ms	54.5 ± 4.8	59.5 ± 2.4	46.5 ± 1.2	**59.7** ± 3.4
antmaze-ls	39.2 ± 5.9	**45.1** ± 5.9	0.0 ± 0.0	24.0 ± 11.0
antmaze-me	15.1 ± 5.5	**29.2** ± 7.0	0.0 ± 0.0	22.8 ± 6.5
cube-single	20.3 ± 2.9	**55.9** ± 3.0	21.5 ± 0.8	21.6 ± 1.8
cube-double	1.7 ± 0.5	2.1 ± 1.2	**4.7** ± 1.0	0.8 ± 0.8
scene	37.9 ± 3.2	**54.0** ± 6.8	11.5 ± 8.7	40.9 ± 2.0
puzzle-3×3	11.2 ± 0.4	**14.1** ± 2.1	10.9 ± 3.7	10.8 ± 3.0
Average	32.6	**45.3**	12.8	32.0

Evaluation details. Unless otherwise stated, all reported results are averaged over three random seeds. On OGBench, we report success rate; on DMC, episode return; and on HumEnv, expert-normalized return. The five official FB-CPR checkpoints used in the checkpoint-robustness experiment constitute a separate source of variation: they share the same architecture and training recipe and differ only in their pretraining seeds. For this experiment, the reported mean and standard deviation are computed across the five checkpoint-level aggregate scores.

Goal-conditioned RL comparison. For the comparison with goal-conditioned RL (GCRL) in Section 5, we evaluate all methods on the same nine proprioceptive OGBench domains, each containing five evaluation tasks. The zero-shot methods and GCRL use HILP as their representation framework but are trained separately according to their respective objectives. GCRL is trained directly with goal-conditioned supervision and receives the target goal state at evaluation time, whereas OLS and BLS infer a task vector from reward-labeled samples. The main GCRL baseline follows its standard goal-conditioned training configuration and is trained from scratch for each domain and random seed using the full offline training dataset. Each model is trained for 1M gradient steps, and we report only the final checkpoint rather than selecting the best intermediate checkpoint. We use three independent training seeds and evaluate each checkpoint deterministically over 50 episodes for each of the five tasks, using the OGBench success metric. For OLS and BLS, we use the same fixed HILP inference configuration as in the main OGBench experiments.

C Additional Experimental Results

C.1 Detailed Comparison with Baselines on OGBench

We report the per-representation results underlying the baseline comparison in Tables 5 and 6. ReLA and LoLA use 20 reward-labeled interaction episodes per task, whereas OLS and BLS require no task-specific interaction. ZOL is reported only with FB, for which it is designed.

Table 6: OGBench success rates (%) with the FB representation, including the FB-specific ZOL baseline and the online adaptation baselines ReLA and LoLA.

Task	OLS	ZOL	BLS@OLS	BLS@ZOL	ReLA	LoLA
antmaze-mn	77.5 ± 2.6	84.5 ± 1.6	**88.8** ± 4.2	84.4 ± 4.5	70.9 ± 3.0	80.5 ± 2.4
antmaze-ln	52.4 ± 4.0	41.5 ± 0.8	63.5 ± 9.2	**68.7** ± 2.4	39.7 ± 3.1	52.8 ± 5.8
antmaze-ms	69.9 ± 1.7	51.9 ± 1.2	**79.6** ± 2.1	68.1 ± 3.6	76.1 ± 1.9	70.0 ± 2.2
antmaze-ls	51.5 ± 2.9	37.9 ± 5.1	45.7 ± 4.6	36.0 ± 4.6	29.2 ± 1.1	**53.7** ± 3.8
antmaze-me	**50.3** ± 2.0	49.7 ± 3.1	49.2 ± 4.9	48.8 ± 1.4	45.5 ± 0.5	47.6 ± 7.3
cube-single	50.4 ± 0.8	46.4 ± 6.8	54.0 ± 1.8	43.7 ± 6.5	**54.5** ± 2.9	**54.5** ± 3.6
cube-double	1.9 ± 1.0	2.8 ± 0.0	**3.1** ± 1.2	1.7 ± 1.0	2.4 ± 0.7	2.3 ± 0.8
scene	7.6 ± 0.7	33.5 ± 7.9	9.6 ± 0.4	**35.3** ± 8.4	15.2 ± 1.4	8.1 ± 1.2
puzzle-3×3	5.7 ± 1.2	13.7 ± 1.2	9.6 ± 1.8	11.7 ± 1.6	**15.3** ± 2.3	4.9 ± 3.0
Average	40.8	40.2	**44.8**	44.3	38.8	41.6

Table 7: Task-level DMC episode return under the trust-region configuration used in the main experiments. Each cell reports OLS / BLS, averaged over three random seeds. The final column averages over the four feature representations used in Figure 3.

Task	ICVF	HILP	FB	TD-JEPA	Avg.
walker-stand	988 / 961	954 / 959	938 / 944	956 / 966	959.0 / 957.5
walker-walk	850 / 884	943 / 932	921 / 919	754 / 834	867.0 / **892.2**
walker-run	307 / 279	332 / 346	353 / 333	316 / 337	327.0 / 323.8
walker-spin	960 / 972	982 / 968	978 / 977	954 / 962	968.5 / **969.8**
cheetah-walk	979 / 979	255 / 651	876 / 974	967 / 983	769.2 / **896.8**
cheetah-walk-backward	986 / 968	874 / 969	985 / 963	981 / 982	956.5 / **970.5**
cheetah-run	275 / 284	86 / 161	317 / 339	367 / 382	261.2 / **291.5**
cheetah-run-backward	301 / 311	196 / 257	317 / 304	362 / 364	294.0 / **309.0**
quadruped-stand	971 / 974	962 / 961	597 / 638	971 / 962	875.2 / **883.8**
quadruped-walk	501 / 524	561 / 653	482 / 467	535 / 571	519.8 / **553.8**
quadruped-run	489 / 491	489 / 505	335 / 340	442 / 508	438.8 / **461.0**
quadruped-jump	760 / 763	777 / 776	520 / 523	737 / 825	698.5 / **721.8**
pointmass-reach-top-left	919 / 939	961 / 965	823 / 819	615 / 671	829.5 / **848.5**
pointmass-reach-top-right	87 / 710	145 / 141	828 / 801	498 / 545	389.5 / **549.2**
pointmass-reach-bottom-left	486 / 738	687 / 680	827 / 826	573 / 670	643.2 / **728.5**
pointmass-reach-bottom-right	402 / 565	2 / 0	131 / 105	192 / 74	181.8 / **186.0**
pointmass-reach-bottom-left-long	212 / 274	4 / 201	-43 / 85	161 / 377	83.5 / **234.2**
pointmass-loop	788 / 809	346 / 549	354 / 615	703 / 784	547.8 / **689.2**
pointmass-square	181 / 612	488 / 481	397 / 862	778 / 826	461.0 / **695.2**
pointmass-fast-slow	195 / 198	52 / 58	121 / 133	413 / 460	195.2 / **212.2**
Average	582 / **662**	505 / **561**	553 / **599**	614 / **654**	563.3 / **618.7**

C.2 Task-Level Results on DMC

Table 7 reports the task-level results underlying Figure 3. Each cell reports OLS / BLS episode return. The final column averages over ICVF, HILP, FB, and TD-JEPA for each task. Under this task-wise aggregation, BLS improves 18 of the 20 DMC tasks.

C.3 Refinement from a Strong Task-Vector Anchor

We study whether BLS can further improve a task vector that already retrieves a strong zero-shot policy and is obtained independently of OLS. We conduct experiments on 50 goal-reaching tasks in HumEnv using FB-CPR representations (Tirinzoni et al., 2025). We compare against four baselines: OLS, Ψ_G, WR, and $B(g)$, where $B(g)$ directly uses the backward embedding of the goal state as the task vector. Following Tirinzoni et al. (2025), we define weighted reward inference (WR) as $z_{\mathrm{WR}} \propto \mathbb{E}_{s\sim\mathcal{D}}\left[e^{10r(s)}B(s)r(s)\right]$. We evaluate performance using three metrics: success rate,

Table 8: Goal-reaching on HumEnv (50 target poses, FB-CPR).

	OLS	Ψ_G	WR	$B(g)$	BLS$_{@B(g)}$
success ↑	0.144 ± 0.020	0.164 ± 0.008	0.148 ± 0.032	0.500 ± 0.022	**0.516** ± 0.032
proximity ↑	0.455 ± 0.023	0.460 ± 0.022	0.453 ± 0.022	0.681 ± 0.009	**0.699** ± 0.009
distance ↓	3.559 ± 0.116	3.309 ± 0.116	3.533 ± 0.094	2.668 ± 0.029	**2.630** ± 0.038

Table 9: Expert-normalized return on HumEnv across five official FB-CPR checkpoints.

Checkpoint	WR	Ψ_G	OLS	BLS$_{@OLS}$	BLS$_{@WR}$
S-1	0.677	0.673	0.509	0.686	**0.744**
S-2	0.575	0.573	0.491	0.638	**0.696**
S-3	0.575	0.560	0.480	0.641	**0.696**
S-4	0.624	0.622	0.524	0.655	**0.679**
S-5	0.615	0.640	0.477	0.622	**0.686**
Mean	0.613 ± 0.038	0.614 ± 0.042	0.496 ± 0.018	0.648 ± 0.022	**0.700** ± 0.023

proximity, which measures the average fraction of time the agent maintains the goal pose, and *distance*, which measures the distance to the target pose.

As shown in Table 8, $B(g)$ is the strongest baseline, achieving a success rate of 0.500, substantially outperforming the second-best baseline, Ψ_G, at 0.164. We therefore use $B(g)$ as the anchor for BLS, denoted as BLS$_{@B(g)}$, with a fixed $\lambda = 20$. Despite starting from this strong anchor, BLS consistently improves all three metrics: success increases from 0.500 to 0.516, proximity from 0.681 to 0.699, and distance decreases from 2.668 to 2.630. These results show that BLS is not limited to correcting OLS and can further refine a strong task vector obtained from a different inference mechanism.

C.4 Results across FB-CPR Checkpoints

We evaluate BLS on the five official FB-CPR checkpoints to examine whether its improvements are robust to variations in representation pretraining. These checkpoints share the same architecture and training recipe but are trained with different random seeds. For each checkpoint, we evaluate all 45 HumEnv tasks and compare against OLS, weighted regression (WR), and Ψ_G, where Ψ_G computes the task vector by averaging the backward embeddings over the goal set G. We apply BLS using either OLS or WR as the anchor, denoted as BLS$_{@OLS}$ and BLS$_{@WR}$, respectively. Table 9 reports the individual results for all five checkpoints.

As shown in Table 9, BLS improves both anchors across the five checkpoints. With OLS as the anchor, the average expert-normalized return increases from 0.496 to 0.648; with WR as the anchor, it increases from 0.613 to 0.700. This shows that the benefit of BLS is robust across independently trained FB-CPR checkpoints rather than being specific to a particular pretrained representation.